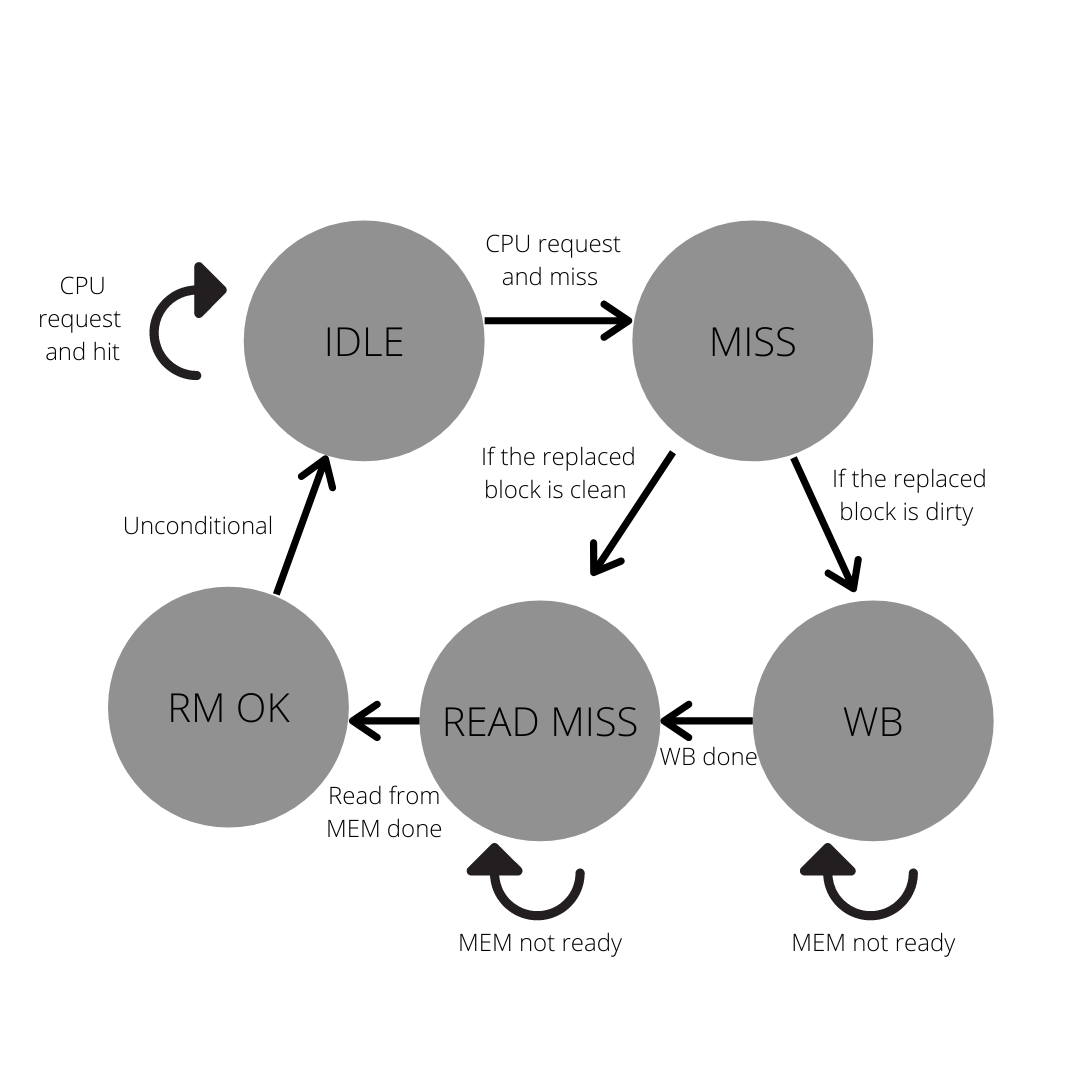
Report for CA Project2 b06611048

1. Modules Explanation cache controller FSM



* 1. dcache\_controller.v: The Bridge to connect CPU, cache, and data memory. In its interface to CPU, it has the address and data that CPU requests or sends as inputs along with read/write control signals, and it can output data (4bytes) that CPU requests and send out control signal to stall the pipeline of CPU. In its interface to data memory, it can send out a whole block of data and request a block to store in SRAM with address, and also enable and write control signals. The inputs in data memory are block data input and mem\_ack, which can notify the controller that the operation on data memory is done. The third interface is to the dcache\_sram. The controller sends the tag field, index field of the address along with enable, and write control signals. It can also send data block to cache if CPU writes data. The cache controller receives hit signal, data block in cache, and tag of data, so it can know if a data block is hit and dirty. The entire controller is a FSM, and the diagram is in the picture. In every state, some control signals change. There four control signals controlled by the FSM, which are mem\_enable, mem\_write, cache\_write, write\_back. write\_back controls the address send to the data memory. If it is on it will send the address of the data in cache. In idle state, all four is off. In miss state, all four is off as well but we can know if the replaced block is dirty or not. In write back stage, mem\_write, mem\_enable, write\_back is on. In read\_miss state, cache\_write and mem\_enable are on while others off. In read miss ok state cache\_write is on and other three off. And back to idle, all four are off. Another control signal, cache\_enable, is controlled by CPU request, which is either read or write. As for the part of sending requested data to CPU, the controller, if data hit, will send 4 byte data according to the offset field and the data block from sram. Additionally on CPU write, the controller will first read the data block, and then write the word and send the block back to cache.
  2. dcache\_sram.v: 2-way associative, has clk\_i, rst\_i, enable, write, tag, data, addr (index), data block in as inputs. The outputs are hit, data, and tag. Upon CPU request, the sram will go to the data slots according to index and determine hit/miss via comparing tags and valid bits. In a miss or CPU write, the sram will know if it’s a CPU write or miss. In a CPU write, it will write on the data block in sram in a write hit instead of write on a copy stored in the other associate block. In a miss, it will read data block and write according to LRU replacement.
  3. PC.v, IFID.v, IDEX.v, EXMEM.v, MEMWB.v: Add an extra control signal, stall, from cache controller to stall the pipeline of miss to wait for memory access.
  4. Testbench.v: Data memory initialization, pipeline regs initialization. Change [23:0] tag to [24:0] tag, since it should be 25-bit (tag+dirty+valid).

1. Difficulties Encountered and Solutions in This Project

* In a CPU write, instead of writing on the target block in cache, it first read the block, and after write it store the block in the same index but different associative slot due to LRU replacement.

Solution: sram now knows if it’s a normal miss or a CPU write. On a CPU write, it will replace the write request block with new written value.

* The flush operation bugged. It did not wrote dirty block back to memory with the flush operation FLUSH IF DIRTY AND VALID.

Solution: Founded out that in TA’s code in testbench.v, the [23:0] tag is 1 bit shy from 25-bit. As a result, there were no valid bits so flush will not store anything.

1. Development Environment

* OS: Windows 10
* Compiler: iverilog
* Text editing: VS Code